FastForward for Efficient Pipeline Parallelism

John Giacomoni, Tipp Moseley, and Manish Vachharajani
University of Colorado at Boulder

Abstract

High-rate core-to-core communication is critical for efficient pipeline-parallel software architectures. This paper introduces FastForward, a software-only low-overhead high-rate queue algorithm for pipeline parallelism on multicore architectures. FastForward uses an architecturally-tuned domain-specific adaptation of concurrent lock-free queues to provide low-latency and low-overhead core-to-core communication. Enqueue and dequeue times on a 2 GHz Opteron 270 based system are as low as 36 ns, up to 4x faster than Lamport's solution.

1 Design and Initial Results

Traditionally, improvements in processor design and fabrication technology have permitted software developers to deliver next generation applications, including modern genomics and software define radios. The challenge with multicore systems is to develop a set of techniques or hardware modifications that help application level developers continue to harness the power of systems.

This work introduces a new concurrent lock-free single-producer/single-consumer queue algorithm (FastForward) that is up to 4x faster than Lamport's queue [2] on commodity cache-coherent processors, permitting developers to achieve performance improvements with fine-grain parallelism [1]. FastForward does this by eliminating the cache-unfriendly implicit coupling in Lamport's queue.

Observe that alternating queue operations with Lamport's algorithm (Figure 1(a)) necessitate the transfer of the head and tail indices between caches for every operation. FastForward counter-intuitively decouples operation by coupling control and data transfer into the storage buffer (Figure 1(b)). Decoupled operation is ensured by separating the producer and consumer in time, thus permitting each processor to operate concurrently on separate cache lines without interference. Hardware prefetching masks the cost of cache transfers for the storage array itself.

The references [1] prove that "in the program order of the consumer, the consumer dequeues values in the same order that they were enqueued in the producer's program order," for strong to weakly ordered consistency models, showing that FastForward works even on relaxed memory models.

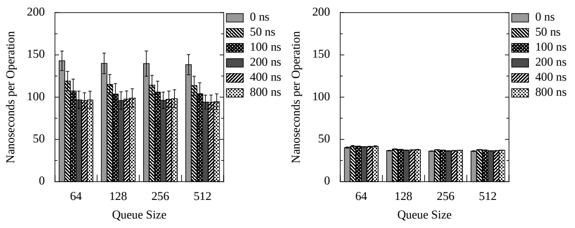
```
if(NEXT(head) == tail){
  // Handle full queue.
}
buf[head] = data;
head = NEXT(head);
```

(a) Lamport

```
if(NULL != buf[head]){
  // Handle full queue.
}
buf[head] = data;
head = NEXT(head);
```

(b) FastForward

Figure 1. Enqueue operation pseudo code.



(a) Lamport (b) FastForward

Figure 2. Performance Comparison.

Figure 2 shows that FastForward outperforms Lamport's queue, for pipeline parallelism, on a 2GHz AMD Opteron 270 with 4 cores by up to 4x while being invariant with respect to both queue size and "work" time between queue operations. Additional results demonstrate equivalent performance across dies, with memory fences for pointer payloads, and highlight the prefetch unit's contribution [1].

In conclusion, FastForward allows one to develop efficient fine-grain pipeline parallel applications on commodity cache-coherent architectures, without hardware modifications. Additionally, the decoupling techniques used in FastForward can be applied to other parallel organizations if there is sufficient data to ensure decoupled operation.

References

[1] J. Giacomoni, T. Moseley, and M. Vachharajani. FastForward for efficient pipeline parallelism. Technical Report CU-CS-1028-07, Univerity of Colorado at Boulder, 2007.

[2] L. Lamport. Specifying concurrent program modules. *ACM Trans. Program. Lang. Syst.*, 5(2):190–222, 1983.